DOCKET NO.: MSFT-2849 / 306818.01
Application No.: 10/692,350
Office Action Dated: August 31, 2006

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:
Conor J. Cunningham *et al.*

Application No.: 10/692,350

Filing Date: October 23, 2003

For: TYPE PATH INDEXING

Confirmation No.: 8548

Group Art Unit: 2162

Examiner: Truong, Cam Y T

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION PURSUANT TO 37 C.F.R. § 1.131

I, Conor J. Cunningham, declare and say that:

1. I am one of the named inventors of the invention described and claimed in U.S. Patent Application No. 10/692,350 (the above-identified application, hereinafter "the '350 application"), which was filed with the United States Patent and Trademark Office on October 23, 2002.

2. As an inventor, I am familiar with the '350 application and the associated rejection alleged in the outstanding Official Action, dated August 31, 2006. I am also familiar with the publication cited by the United States Patent and Trademark Office (the publication being U.S. Pub. No. 2005/0055355 to Murthy *et al.*, hereafter referred to as "the Murthy application") in connection with the outstanding Official Action.

3. In particular, I understand that claims 1, 3-8, and 10-14 were rejected under 35 U.S.C. § 103(a) as being allegedly obvious over the Murthy application in view of U.S Pat. No. 6,591,260 to Schwarzhoff *et al.* (hereafter "Schwarzhoff"). I also understand that claims 1-14 were rejected under 35 U.S.C. § 103(a) as being allegedly obvious over U.S. Pat. No. 6,016,497 to Suver in view of the Murthy application in further view of Schwarzhoff. Furthermore, I understand that claims 1-14 were rejected under 35 U.S.C. § 103(a) as being allegedly obvious over U.S. Pat. No. 6,016,497 to Suver in view of the Murthy application in

further view of U.S. Pub. No. 2004/0068696 to Seyrat *et al.* (hereafter "Seyrat"). Finally, I understand that claims 2 and 9 were rejected under 35 U.S.C. § 103(a) as being allegedly obvious over the Murthy application in view of Schwarzhoff in further view of U.S. Pat. No. 6,643,633 to Chau *et al.*

4. I understand that the Murthy application was published on March 10, 2005, was filed on January 23, 2004, and claims priority to Provisional Application No. 60/500,450, which was filed on September 5, 2003.

5. In accordance with 37 CFR § 1.131, as an inventor of the subject matter of the rejected claims, and without conceding the propriety of the outstanding rejections, I hereby declare that the other inventors and I invented the subject matter of the rejected claims prior to September 5, 2003, the earliest possible effective date of the Murthy application, and thus the other inventors and I are prior inventors as referred to in that Section.

6. Prior to September 5, 2003, the other inventors and I invented the subject of the '350 application. This is demonstrated by the initial internal proposal document dated July 9, 2003 (attached hereto as Appendix A) and the final functional specification document dated September 3, 2003 (attached hereto as Appendix B). The internal proposal document was created to explain the ideas to others on the development team.

7. Accordingly, it is my belief that the Appendices attached hereto evidences my and the other inventors' possession of the invention described in the '350 application prior to September 5, 2003, thereby removing the Murthy application as an applicable reference.

8. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information or belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or by imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful statements may jeopardize the validity of the application, any patent issuing there upon, or any patent to which this verified statement is directed.

Date: November 22, 2006

Conor J. Cunningham

An Indexing Proposal for Efficient UDT Type Hierarchy Retrieval

Conor Cunningham, Venkatesh Ramachandran

1. Introduction

WinFS's schema model poses some new challenges to the SQL Query Processor. UDTs are used extensively, and it is common to retrieve all UDTs from a table based on the UDT type. Furthermore, WinFS uses UDT inheritance, and it is also a requirement to retrieve all elements of a given type and also any subtype from a table. Multiple tables exist, each containing a different number of UDTs, types, type topology, and UDT distribution within that topology. These properties make it difficult to make accurate cardinality and cost estimates, and it also makes it difficult to efficiently retrieve values based on type/type hierarchy.

2. The Solution – A "Type Index"

Existing indexing infrastructure can be used to solve this problem. B-Tree indexes have the ability to *seek* to a particular location and retrieve one value or a series of keys with the same value. Furthermore, it is also possible to seek on a prefix of a key and return a *range* of values all containing the same prefix. These properties can be leveraged to allow efficient retrieval based on either type or type hierarchy.

Each UDT type is assumed to be a small (perhaps 4-byte) fixed-length value. Furthermore, a *hierarchical type-id* can also be defined as the concatenation of the parent *type-id*s into a string or binary field. If each row containing a UDT also has a computed column representing this hierarchical encoding, an index can be created over this new column. Operations to return all UDTs in a given type hierarchy can be solved using a range prefix operation, and exact matches on the type-id can be used to solve non-hierarchical matches.

This encoding has an additional side-benefit – it is possible to use a histogram on this binary encoding to determine statistical information about the distribution of UDTs in the type hierarchy. Most type hierarchies will have far fewer members than the number of steps (200) in our histograms, and the depth of these hierarchies should be far less than the total number of bytes of prefix data (100) used when building those histogram steps. By walking the histogram during selectivity estimation with knowledge of this hierarchical encoding, a fairly accurate cardinality estimate can be generated for use in the query. (Design note: If an intrinsic function is used to estimate the selectivity per the algorithm described, it should be possible to remove the CLR call to ISOF completely and just do the cardinality estimate on the intrinsic function)

3. Necessary Work

1. New built-in functions need to be added to return a type-id from a UDT and to return a hierarchical type-id encoding from a type-id.

2. While the B-Tree indexes support prefix-matching for string types, this capability apparently does not yet exist throughout the product for binary types. Adding this capability would simplify the introduction of this feature.
3. WinFS needs to create a computed column on every UDT-hosting table (i.e. all item tables)
4. The ISOF scalar needs to be Algebrized to contain an additional/replacement predicate. One suggested technique is to use the UDT valref to find the base table containing the UDT. Base table computed columns would then be used to determine the existence of a computed column comprising of a scalar expression over the two new built-ins listed in #1. If such a column is found, an implied predicate would be added using the built-ins from #1. Another suggested technique uses a portion of the "implication rule" framework that was cut from Yukon. Either technique is fine, cost permitting.
5. The Query Optimizer will write new cardinality/selectivity estimation code for the ISOF scalar (index selection should be automatic).

4. Extra Details

The initial review of this proposal suggested that we use the "Tries" feature in the optimizer to estimate selectivity. I discussed the issue with Campbell, and he suggested that it is not any better than a histogram for the scenario we are trying to solve. The interesting details:

- Tries support 40 character of prefix and 40 characters of suffix information – this is eventually going to be fixed to match histograms (100 each)
- Tries have a lot of logic for estimating the cardinality of a specific string pattern, but do not estimate ranges. At each node in the Tries tree, only the number of exact matches for that string is stored. Generating an estimate for all strings with that prefix is not a Tries feature (currently)
- Tries was originally invented to get better estimates for a larger number of distinct constant strings beyond what histograms stored. However, in our scenario, we're not likely to see more estimates than the number of steps possible in the histogram (200)
- Histograms are better at estimating range information – the prefix matching that we'd like to do in this example is essentially a range prefix estimation
- The histogram will cluster results together based on hierarchy (in this encoding). The tree structure may not be collocated. This may not matter if everything is in memory

Functional Spec for a Built-in Function To Support Indexing and Fast Retrieval on the Type of UDT Values in SQL Server

Author: {**Eric Hanson (ehans), Conor Cunningham (conorc)**}

Category: {UDTs }

Date 9/3/2003 11:19 AM

Draft 1.4

Distribution: Microsoft Internal Distribution

1 Abstract and Context

We propose an external interface for a built-in function that provides *hierarchical type ids* for UDTs. A hierarchical type id is a varbinary value that uniquely identifies the type of *UDT_expression* within a type hierarchy. By using this function in queries and in the creation of appropriate computed column indexes, you can support efficient UDT type hierarchy retrieval as outlined in [Cun03].

2 Requirements, Background, Assumptions and Restrictions

Support fast retrieval of all values in a table with a specific type, or all values whose type is in the tree rooted at a given type in the type hierarchy.

Hierarchical type id values [Cun03] must be short to reduce the amount of data that must be stored in an index.

Hierarchical type ids should be less than 100 characters long at most for all types in the WinFS schema, so we get the best possible statistics on type path columns.

The function to retrieve a hierarchical type id given an internal type id must be fast because it is used during index creation.

We assume that you can't drop a type while instances of it remain in the database.[1] Otherwise, index entries containing the hierarchical type id of a dropped type might remain in indexes, but it would not be possible to interpret them. Furthermore, adding a new type after dropping one might result in a hierarchical type id that was the same as that of an existing instance, causing an error.

3 Description

The system supports the following publicly visible function:

[1] The current plan as of 8/28/03, according to Jose Blakely, is that you can't drop a type if it or any of its ancestors are used in the definition of a table that still exists. This validates this assumption.

Function	Parameters	Result
HIERARCHICAL _TYPE_ID	(*UDT_expression*)	A varchar value containing the concatenation of type ids of types along the path from the root of the type hierarchy to the most specific type of *UDT_expression*. The result returned has binary collation Latin1_General_BIN. This varchar value can contain non-printing characters. Cast it to varbinary if you wish to display the values. The format of hierarchical type id values is undocumented. You should not depend on their internal structure in your application.

UDT_expression can be any user-defined-type-valued expression. The HIERARCHICAL_TYPE_ID function is DETERMINISTIC and PRECISE. This allows you to build indexes on computed columns derived using it. You do not have to make these computed columns PERSISTED.

To determine if a type path is a prefix of another, the system internally uses a HAS_PREFIX operator that is built using the existing LIKE prefix-scanning capability. This prefix scanning capability is now used to implement matching for conditions of the form "character_expression LIKE 'prefix_string%'" where prefix_string is some constant string that does not contain any wildcard characters such as %. This HAS_PREFIX operator is not user-visible. HAS_PREFIX can use an index scan if an index is available.[2]

3.1 Internal Hierarchical Type ID Format

The internal format of a hierarchical type ID shall be a varbinary value whose length is a multiple of 4 bytes, containing the concatenation of type ids (stored internally as 4-byte integers) of types along the path from the root of the type hierarchy to the most specific type of UDT_expression.

Note to documentation team: don't document this in user documentation.

Note to developer(s): you can use a more compact format if time allows, but don't use a less compact format. See for example the compressed format described in the pick list (section 4.1).

3.2 Indexing Values with a Given Type for Fast Search

You may have a large table with an attribute whose values are of a user-defined type. In this case, you may want to support fast retrieval of rows where values of that attribute have a specific type, or are a subtype of a given type. To support fast search in this situation, create an index on a computed column created using HIERARCHICAL_TYPE_ID.

Assume that there is the following type hierarchy of UDTs:

```
CREATE TYPE person_t EXTERNAL NAME [asm]:[Person]
CREATE TYPE employee_t EXTERNAL NAME [asm]:[Employee] UNDER person_t
CREATE TYPE hourly_employee_t EXTERNAL NAME [asm]:[HourlyEmployee]
  UNDER employee_t
CREATE TYPE salaried_employee_t EXTERNAL NAME [asm]:[SalariedEmployee]
  UNDER employee_t
```

Furthermore, there is a table defined as follows:

```
CREATE TABLE person(pcol person_t)
```

[2] Contact Ramachandran Venkatesh (VenkaR) for details regarding extensions to support prefix matching for varbinary fields using the LIKE infrastructure.

To enable fast lookup of person rows while filtering on the type of pcol using the IS OF operator, add a computed column and index on the hierarchical type id for person.pcol of the person table as follows:

ALTER TABLE person ADD pcol_htid AS HIERARCHICAL_TYPE_ID(pcol)

CREATE INDEX person_htid_idx ON person(pcol_htid)

This will, for example, allow you to rapidly find all information about hourly employees even if the "person" table is large.

Create a CLUSTERED index on HIERARCHICAL_TYPE_ID if fast retrieval of all values of a given type is a high priority for your application. Making the index CLUSTERED will group values of the same type, or in the same subtree of the type hierarchy, on the same page or on nearby pages on disk.

3.3 Query rewrite

Note to documentation team: do not include discussion of query rewrite in user documentation. Instead, we recommend that you include a general statement that IS OF predicates are implemented internally via the HIERARCHICAL_TYPE_ID, and that an index on a computed column derived from HIERARCHICAL_TYPE_ID can help speed up queries on type that use the IS OF operator.

The system uses query-rewrite internally to process IS [NOT] OF predicates, also known as *type predicates* [ISO99]. The format of a type predicate is defined as follows:

Type_predicate ::= UDT_expression IS [NOT] OF (Type_list)

Type_list ::= user_defined_type_specification [, …n]
User_defined_type_specification ::=
    Inclusive_UDT_specification | Exclusive_UDT_specification
Inclusive_UDT_specification ::= UDT_name
Exclusive_UDT_specification ::= ONLY UDT_name

An expression of the form

UDT_expression IS NOT OF (type_list)

is equivalent to

NOT (UDT_expression IS OF (type_list))

Hence, we will not discuss query rewrite for IS NOT OF predicates further.

A type predicate of the form

UDT_expression IS OF (type_list)

is rewritten as a disjunction of predicates testing whether the type of *UDT_expression* matches the entries in *type_list*.

A test to see if UDT_expression IS OF an Inclusive_UDT_specification is expressed in the rewritten query as:

```
HIERARCHICAL_TYPE_ID(UDT_expression) HAS_PREFIX
        <<constant hierarchical type id of UDT_name>>
```

Here, <<constant hierarchical type id of UDT_name>> is a varbinary constant representing the hierarchical type id of UDT_name.

A test to see if UDT_expression IS OF an Exlusive_UDT_specification is expressed in the rewritten query as:

```
HIERARCHICAL_TYPE_ID(UDT_expression) =
        <<constant hierarchical type id of UDT_name>>
```

Important: the HIERARCHICAL_TYPE_ID(UDT_expression) expression in the rewritten conditions above will match an index on a computed column built from the same expression, such as person_htid_idx.

Examples

The following examples show application of the rewrite rules described above to support IS OF and IS OF (ONLY…) predicates.

```
-- Find all persons who are employees of any kind
SELECT * FROM person WHERE pcol IS OF employee_t

-- Rewritten query:
SELECT * FROM person
WHERE HIERARCHICAL_TYPE_ID(pcol)
         HAS_PREFIX <<constant hierarchical type id of employee_t>>

-- Find all persons who are of type employee_t but not one of its subtypes.
SELECT * FROM person WHERE pcol IS OF (ONLY employee_t)

-- Rewritten query:
SELECT * FROM person
WHERE HIERARCHICAL_TYPE_ID(pcol) = <<constant hierarchical type id of employee_t>>

-- Find all persons who are salaried or hourly employees.
SELECT * FROM person WHERE pcol IS OF (hourly_employee_t, salaried_employee_t)

-- Rewritten query:
SELECT * FROM person
WHERE
  (HIERARCHICAL_TYPE_ID(pcol)
        HAS_PREFIX <<constant hierarchical type id of hourly_employee_t>>
  OR
  HIERARCHICAL_TYPE_ID(pcol)
        HAS_PREFIX <<constant hierarchical type id of salaried_employee_t>>)
```

4 Other Areas of Impact

Impacted Area	**Affected by this feature?**
GUI	No
Testability	new tests are required to verify correct behavior of HIERARCHICAL_TYPE_ID and verify correct index matching for IS OF predicates.

Dependency on Others	Depents on implementation of typeid value as 4 byte int allocated internally or by CREATE TYPE.
Dependency from Others	Meets WinFS requirement for fast retrieval based on type of a UDT.
Localization	no
Incompatibilities & Migration	no
Performance	Performance for IS OF predicates will typically improve when an index can be utilized. Performance of updates will be worse in some cases due to presence of an index on hierarchical type id.
Security & Privileges	no
Errors & Warnings	TBD
Competition	Oracle supports similar functionality with its SYS_TYPE function and function-based indexes on SYS_TYPE. They do not support a notion of a hierarchical type ID.
Standards	N/A
Terminology	"Hierarchical type ID" – see section 1
Pick List	see section 4.1
Future Considerations	Any change in the format or value of herarchical type ID from release to release of SQL Server or the .NET Frameworks will be costly because it will require index reconstruction overhead. Changing the format should be avoided unless the benefits outweigh the drawbacks. Alternatives such as creating a new function hierarchical_type_id2() that returns compressed hierarchical type ids (see section 4.1), and deprecating hierarchical_type_id(), should be considered.
Active Issues	N/A
Resolved Issues	N/A

4.1 Pick List

The following are nice to have for completeness to enable reflection scenarios, but are not required for WinFS:

- Add an overloaded version of the function TYPE_ID() to return the built-in type id of a UDT expression.
- Add an overloaded version of the function TYPE_NAME() for a single varbinary hierarchical type id (htid) argument to return the string name for the type with that htid.

  The example below shows how to count the occurrences of each distinct type in a hierarchy in a table.

  ```
  -- Count the number of persons of each type
  SELECT TYPE_NAME(TYPE_ID(pcol)), TYPE_ID(pcol), count(*)
  FROM person
  GROUP BY TYPE_NAME(TYPE_ID(pcol)), TYPE_ID(pcol)
  ```

- Provide a compression scheme for hierarchical type ids. This would reduce the size of index keys size built from hierarchical type ids. For example encode them as base-254 integers (with digits as byte values 0-254 decimal) and use the byte 255 as a separator. Then a hierarchical type id would be of the form:

  <typeid> <separator> <typeid> <separator> … <separator> <typeid>

  In most real-world scenarios, this would result in only 2 or 3 bytes per typeid on the path instead of 4.

- Add type subsumption capability to query rewrite when the type_list in a type predicate has more than one entry. For example,

  UDT_expression IS OF(person_t,employee_t)

  is equivalent to

  UDT_expression IS OF(person_t)

  because employee_t is a subtype of person_t.

5 Implementation Notes

Here are issues to note related to the implementation of this feature.

1. We have assumed that internally, we will use the LIKE operator with to implement the HAS_PREFIX operator. LIKE must be made to use binary collation (Latin1_General_BIN), either by default based on the collation of its inputs, or explicitly with a COLLATE clause. We need to confirm that there are not difficulties implementing this.
2. The index built on the hierarchical_type_id() computed column must have binary collation to match the collation sequence used for LIKE. We must ensure that the collation of the index matches the collation used for the LIKE comparison – both must be binary.
3. Special characters (%, _, [,], ^) can appear in the right-hand argument to LIKE because it is composed of arbitrary binary data. If the standard LIKE pattern-matching code is used to implement HAS_PREFIX, these characters must be escaped in the pattern using an ESCAPE <escape-character> clause on LIKE. Instances of the escape character used must also be escaped.
4. Bytes with value 0 can appear in hierarchical type ids. We've assumed that LIKE and equality comparison will not be affected by this, and done some limited testing that seems to verify this assumption. The assumption must be validated during implementation.

6 Considerations for Other Features, Components, and Products

Area	Owner	PM	Dev/Lead	Test/Lead	Sign-Off
Relational Engine					
RE – Query Processor		LuborK			
RE - T-SQL Lang		BalajiR			
RE – Metadata, Sys. SPs		BalajiR			
RE - Distributed Query		BalajiR			
RE – Partitioning		LuborK			
RE – Filestream		DonV			
RE – Security		GChander			
RE – XML		ShankarP			
Storage Engine					
SE – Util/load/backup		GregSmit			
SE – Core		MirekS			
Books On-line		AlanBr			
CE		BrianSa			

SQL Tools (EM)	RWaymi			
Failover Cluster	DonV			
Full Text Search	ACencini			
MSDE	RGeorge			
Plato	KamalH			
Replication	VaqarP			
Setup	Microthk			
WebData – Managed	PabloCas			
WebData - Non Mnged	BradRhod			
Visual Studio	DanWinn			
Office	BillRamo			

7 Supportability considerations and Notes to PSS

N/A

8 Known Work Outstanding

- Addition of HIERARCHICAL_TYPE_ID function
- Support of prefix-scan on varbinary based on LIKE infrastructure and internal HAS_PREFIX operator
- Implementation of query-rewrite for IS OF predicates.

9 Document Change History

date	Author	draft	purpose of draft
8/14/03	ehans	1.0	First complete draft
8/15/03	ehans	1.1	Modified to accommodate IS OF(type_list) where type_list can have multiple entries.
8/27/03	ehans	1.2	Revised based on feedback from review of Sam Smith and Cliff Dibble's comments.
8/28/03	ehans	1.3	Revised to eliminate need to apply HIEARCICAL_TYPE_ID to a type name directly from T-SQL. Updated assumptions.
9/3/03	ehans	1.4	Added Implementation Notes section with discussion of subtleties regarding LIKE pattern matching, collation, special characters, escape character use, and 0 bytes.

10 References

[Cun03] Conor Cunningham and Venkatesh Ramachandran, *An Indexing Proposal for Efficient UDT Type Hierarchy Retrieval,* July 2003. sql\ntdbms\specs\yukon\winfs\ WinFS-TypeIndexProposal.doc

[ISO99] Database Language SQL—Part 2: Foundation (SQL/Foundation), ISO, Section 8.14, page 320, July 1999.